

*A*rchitecture *D*escription *L*anguages

Languages in Computer Science



GPLs

Java, C, ...



DSLs

SQL, jQuery, ...

GPLs vs DSLs

- Домейн Специфичните Езици (Domain Specific Languages – *DSLs*), са специализирани езици, предназначени за конкретна задача.
- Езици с общо предназначение (General Programming Languages – *GPLs*) като C и Java са създадени за писане на всякакъв вид програма с каквато и да е логика.
- GPLs са по мощни от DSLs, но в замяна, вторите са пригодени към нуждите на конкретен домейн.
- За един DSL не е задължително да представлява изчисления директно или да се стига до програмен код.
 - Може просто да декларира правила, факти и т.н., които могат да бъдат тълкувани по различни начини.

Languages in Computer Science

„Домейн - Специфичните Езици (DSLs) са езици за програмиране с ограничена изразителност, фокусирани върху определен домейн.“



DSLs - examples

```
5  LIBRARY ieee;
6  USE ieee.std_logic_1164.all;
7
8  ENTITY LabExCG4 IS
9  PORT(   u, v, w, x, y : IN BIT;
10         s               : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
11         m : OUT BIT);
12 END LabExCG4;
13
14 ARCHITECTURE Behavior OF LabExCG4 IS
15 BEGIN
16 PROCESS(s)
17 BEGIN
18     CASE s IS
19         WHEN "000" => m <= u;
20         WHEN "001" => m <= v;
21         WHEN "010" => m <= w;
22         WHEN "011" => m <= x;
23         WHEN "100" => m <= y;
24         WHEN OTHERS => m <= y;
25     END CASE;
26 END PROCESS;
27 END Behavior;
```

VHDL

```
SQLQuery1.sql - Ser...Administrator (54)) * X
CREATE TABLE ClientInfoMasked
(
    ClientID int IDENTITY,
    FirstName varchar(65),
    LastName varchar(65),
    PhoneNum bigint
        MASKED WITH (FUNCTION = 'default()'),
    EmailAddr varchar(100)
        MASKED WITH (FUNCTION = 'email()'),
    CreditCardNum varchar(19) MASKED
        WITH (FUNCTION = 'partial(0,"XXXX-XXXX-XXXX-",4)'),
    BirthDT date MASKED
        WITH (FUNCTION = 'default()');

```

SQL

Messages

Command(s) completed successfully.

```
h1 { color: white;
      background: orange;
      border: 1px solid black;
      padding: 0 0 0 0;
      font-weight: bold;
    }
/* begin: seaside-theme */

body {
  background-color: white;
  color: black;
  font-family: Arial, sans-serif;
  margin: 0 4px 0 0;
  border: 12px solid;
}
```

CSS

CSS



AutoCAD

Advantages of DSLs

- ✓ DSL програмите могат да изразяват само важната информация и да крият детайлите.
- ✓ Те са по-кратки, лесни за разбиране, поддръжка и т.н.
- ✓ Могат да се генерират множество артефакти – статистически анализи и т.н.
- ✓ Лесни за използване.
- ✓ Улесняват комуникацията между експертите в домейна (domain experts).

Advantages of DSLs

Simple Example

```
<people>
  <person>
    <name>James</name>
    <surname>Smith</surname>
    <age>50</age>
  </person>
  <person employed="true">
    <name>John</name>
    <surname>Anderson</surname>
    <age>40</age>
  </person>
</people>
```

- Добавя към документите допълнителен синтактичен шум, който разсейва хората.
- Човекът не може лесно и бързо да разбере действителната информация.

Advantages of DSLs

Simple Example

```
<people>
  <person>
    <name>James</name>
    <surname>Smith</surname>
    <age>50</age>
  </person>
  <person employed="true">
    <name>John</name>
    <surname>Anderson</surname>
    <age>40</age>
  </person>
</people>
```



```
person {
  name=James
  surname=Smith
  age=50
}
person employed {
  name=John
  surname=Anderson
  age=40
}
```


Advantages of DSLs

Simple Example

```
<people>
  <person>
    <name>James</name>
    <surname>Smith</surname>
    <age>50</age>
  </person>
  <person employed="true">
    <name>John</name>
    <surname>Anderson</surname>
    <age>40</age>
  </person>
</people>
```



James Smith (50)
John Anderson (40) *employed*

Disadvantages of DSLs

- Допълнителна инвестиция
- Слаба поддръжка на инструменти (пр. редактор)
- Миграцията може да е трудна
- Потребителите трябва да научат нов специфичен език
- Лоша документация за нови DSL (в сравнение с GPL)

Types of DSLs

- *Вътрешен (internal)* или *вграден (embedded)* DSL: проектиран и реализиран с помощта на хост език.
- Може да използва от хост езика граматиката и инструментите.
- Също така обаче важат *ограниченията* на хост езика.
- Предизвикателството с този тип е да се проектира езика така че синтаксисът да е в ограниченията, но да е и достатъчно изразителен.
- По принцип се търси хост език, който да е колкото се може по гъвкав и да има възможно най-малко ограничения.

Types of DSLs

- *Външен (external)* DSL е проектиран да е независим от всеки конкретен език.
- Всеки език и инструмент могат да бъдат използвани за дефиниране на външен DSL (пр. Java и ANTLR).
- Пълна гъвкавост при избора на синтаксиса, символите на езика, операторите, конструкциите и структурата.
- Трябва да се определи граматиката за езика и да се създаде компилатор, който да анализира и обработва синтаксиса и да се грижи за семантиката.
- Един външен DSL предоставя много по голяма гъвкавост, но изисква много повече време за дефинирането му.

Types of DSLs

Internal

External

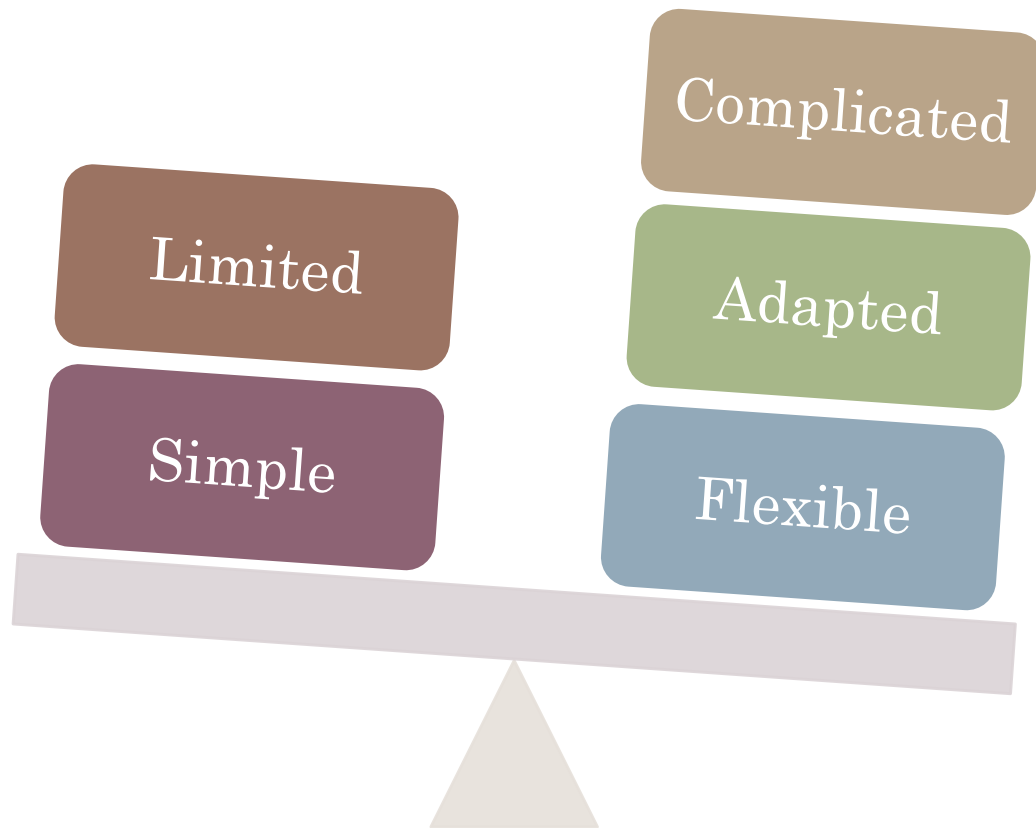
Limited

Simple

Complicated

Adapted

Flexible

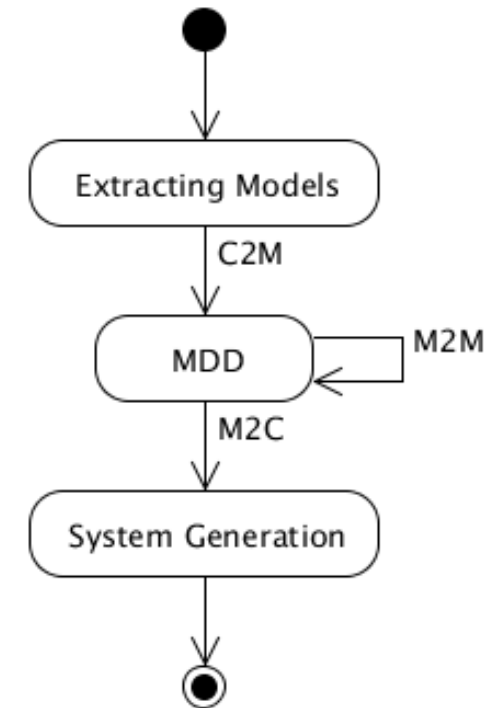


Types of DSLs

Textual

```
digraph Modernization {  
  Start -> ExtractingModels;  
  ExtractingModels -> MDD [label="C2M"];  
  MDD -> MDD [label="M2M"];  
  MDD -> SystemGeneration [label="M2C"];  
  SystemGeneration -> End;  
  
  Start  
    [shape=point,width=0.2,label=""];  
  ExtractingModels  
    [label="Extracting Models",shape=ellipse];  
  MDD  
    [shape=ellipse];  
  SystemGeneration  
    [label="System Generation",shape=ellipse];  
  End  
    [shape=doublecircle,label="",width=0.2,  
      fillcolor=black,style=filled];  
}
```

Graphical

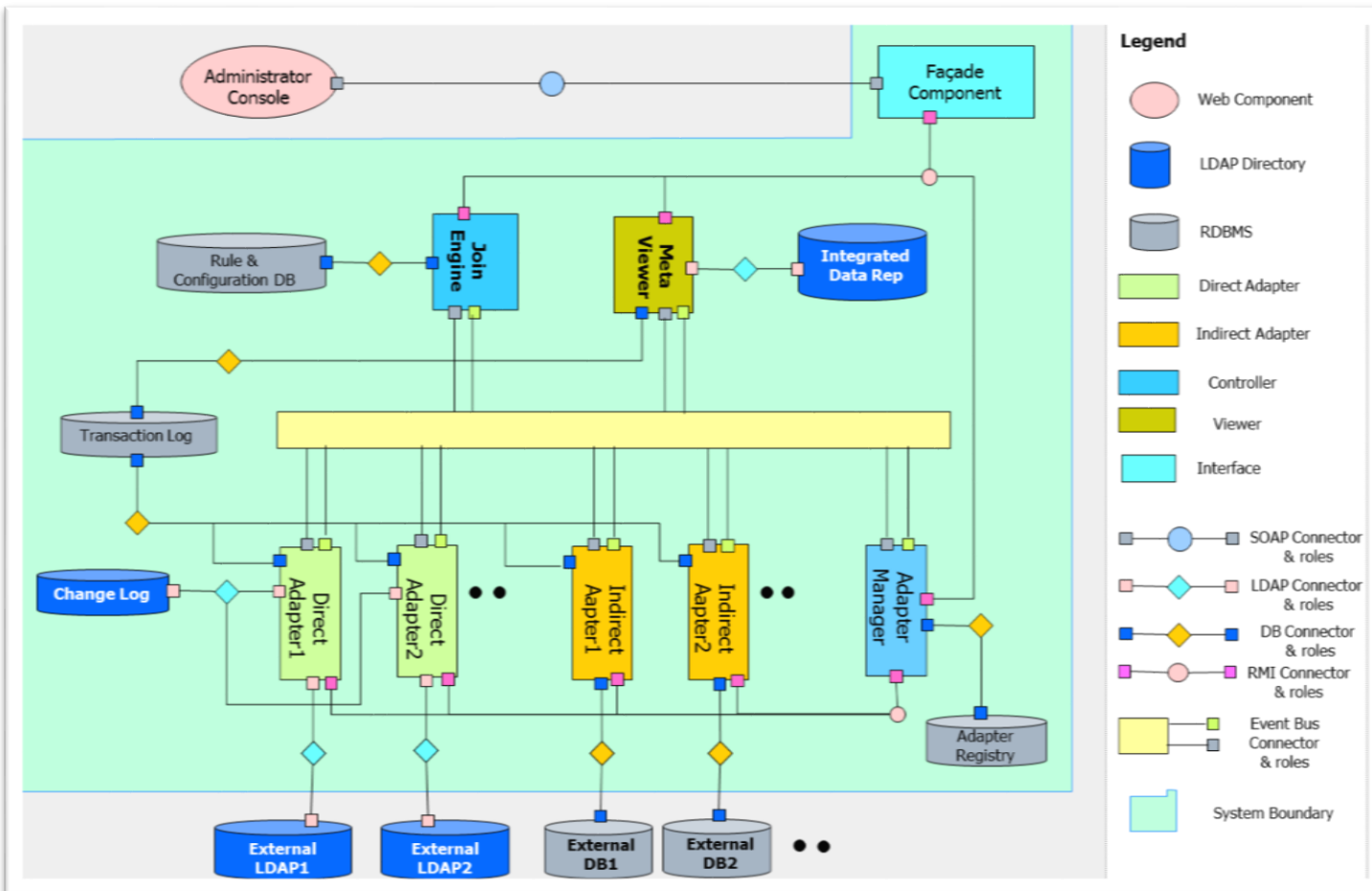


Architecture Description Languages

- *Езиците за Описание на Архитектура (ADLs)* са DSLs, използвани в областта на софтуерните архитектури.
- Формални езици, които се използват за да опишат/представят архитектурата на софтуерни системи.
- Обикновено предоставят конкретен синтаксис за характеризирането на софтуерни архитектури.
- Използването на ADL изисква поддръжка от инструменти за анализиране, визуализиране, анализ и т.н. Тези инструменти (обикновено) са специфични за всеки ADL.

ADLs Representation

ACME Graphical View



ACME Textual View

```

System simple_cs = {
Component client = {
    Port sendRequest;
    Properties { requestRate : float = 17.0;
                  sourceCode : externalFile = "client.c" } }

Component server = {
    Port receiveRequest;
    Properties { idempotent : boolean = true;
                  maxConcurrentClients : integer = 1;
                  multithreaded : boolean = false;
                  sourceCode : externalFile = "server.c" } }

Connector rpc = {
    Role caller;
    Role callee;
    Properties { synchronous : boolean = true;
                  maxRoles : integer = 2;
                  protocol : WrightSpec = "..." } }

Attachments {
    client.send-request to rpc.caller ;
    server.receive-request to rpc.callee }
    
```


ADL categories

Конекторите са важни архитектурни елементи, които един ADL трябва да предоставя. Следователно, важна таксономия по отношение на ADLs е тази, която ги класифицира според тях, предложена от *Amirat* и *Oussalah*:

- *ADLs с неявни конектори*, които не поддържат конектори. ADLs като Darwin и Rapide не считат конекторите за първокласни единици.
- *ADLs с предварително определен набор от конектори*. UniCon е пример за такъв език. Конекторите са предварително дефинирани и вградени в езика.
- *ADLs с явни типове конектори*. Повечето ADLs попадат в тази категория, като разглеждат конекторите като първокласни единици на езика. Изчисленията са описани вътре в компоненти, а конекторите описват механизмите за взаимодействие между тях. Примери за такива езици могат да бъдат открити както в ранни ADLs, като Wright, така и в по-нови, като p-ADL.

Advantages of ADLs

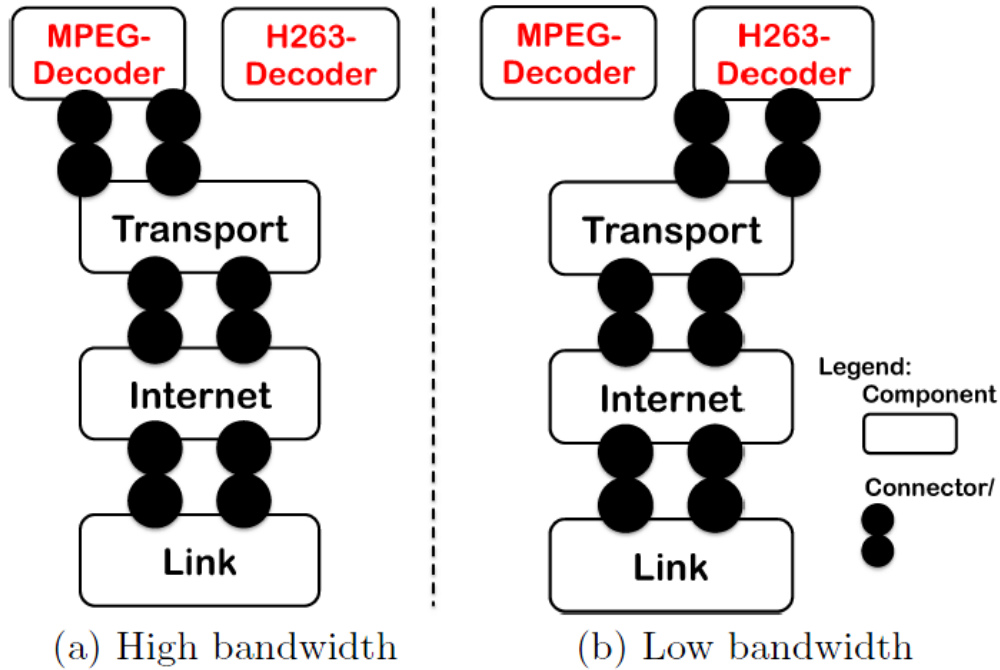
- Предоставят добра основа за комуникацията на архитектурата.
- Лесно могат да бъдат четени както от хора, така и от машини
- Могат да описват една система система на високо ниво
- Предоставят начини за анализ и оценка на архитектурите относно неяснота, производителност, т.н.
- Могат да се използват за автоматично генериране на софтуерни артефакти (модели, code stubs, т.н.)

Disadvantages of ADLs

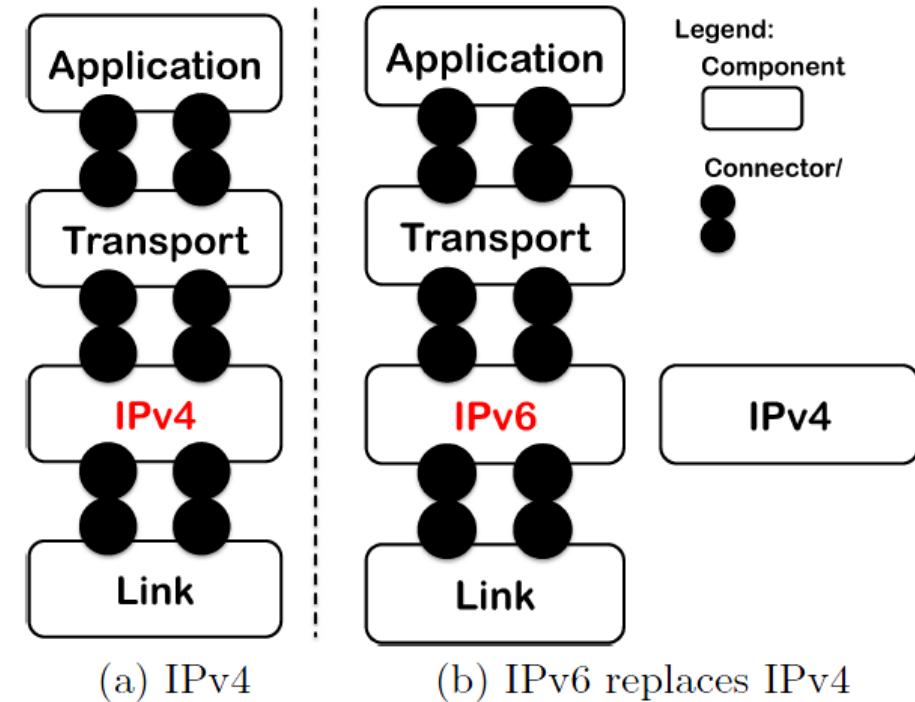
- Високата степен на формалност затруднява интеграцията им в индустриалните жизнени цикли.
- Специализирана семантична основа:
 - Различният анализ изисква различни ADL.
 - Невъзможно е да се изгради ADL, който поддържа всеки вид анализ.
- Ограничена поддръжка на инструменти.
- Много ограничена употреба в индустрията.
- Трудности при изразяването на динамични реконфигурации.

Dynamic Reconfiguration

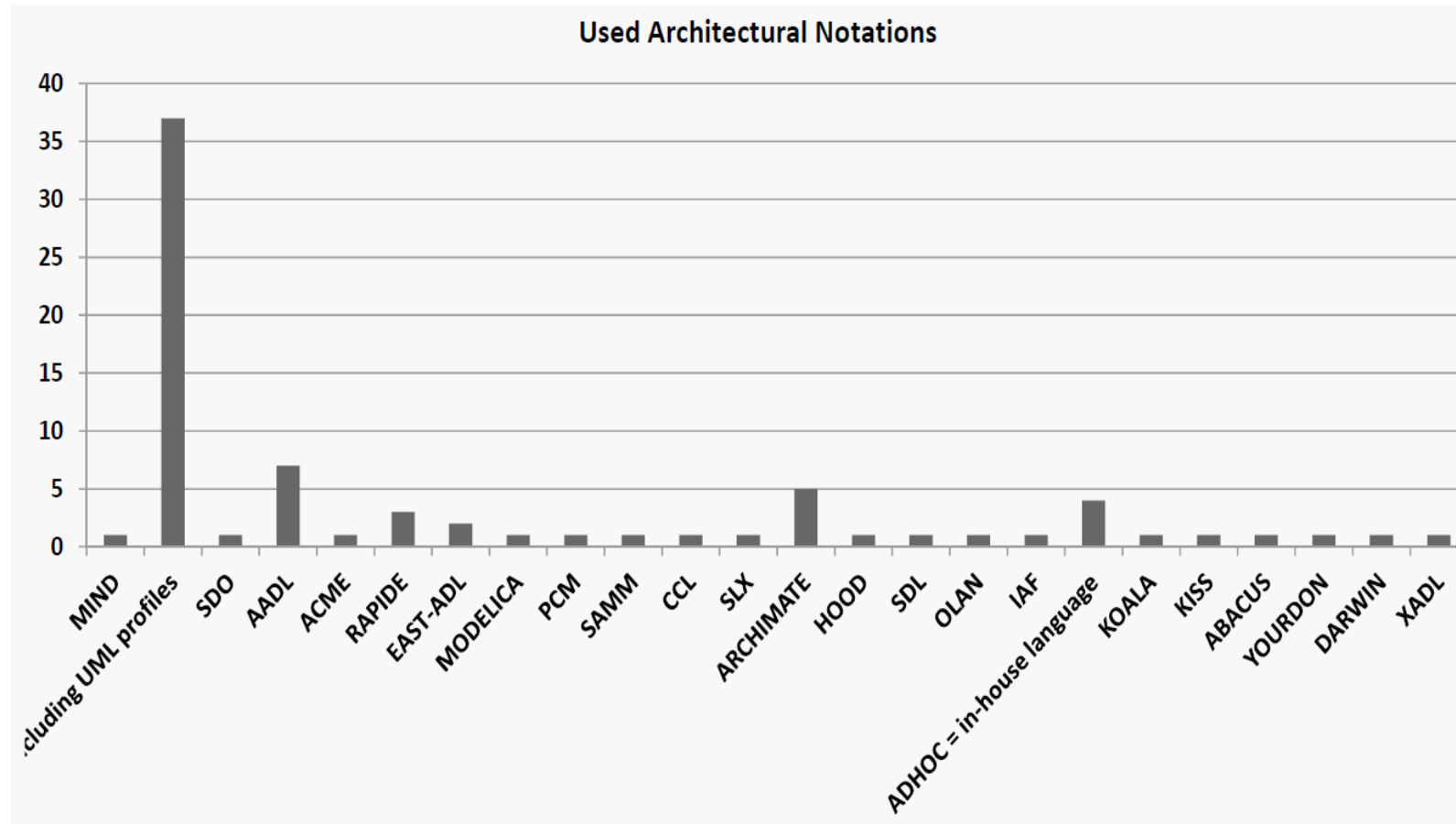
Foreseen



Unforeseen



ADL uses

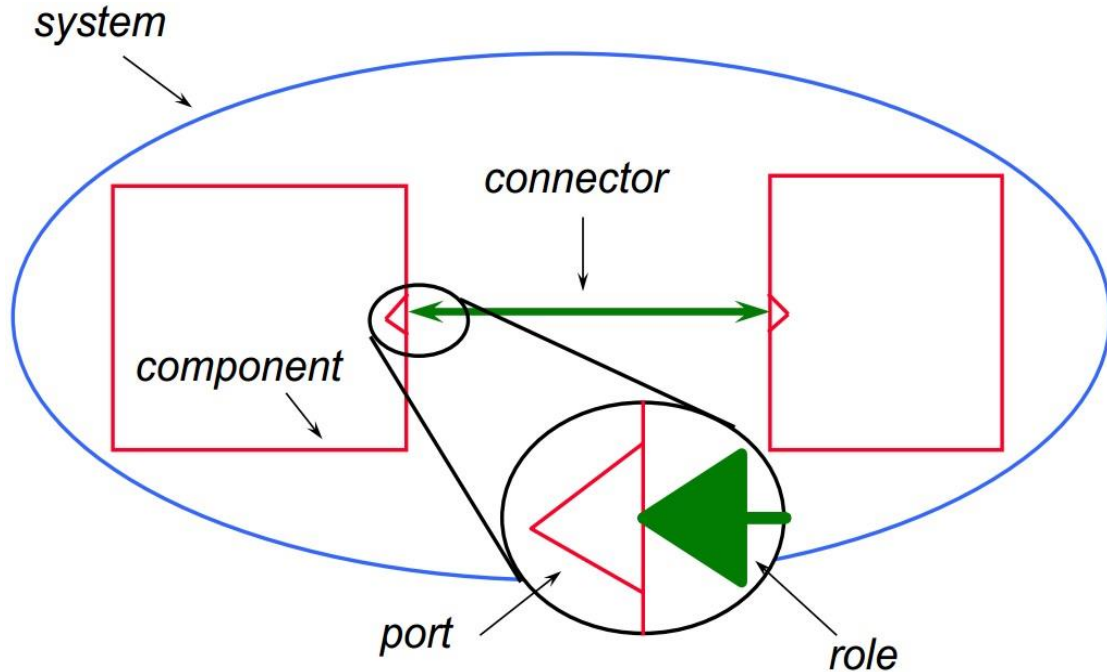


Usage of ADLs in industry

ADLs - ACME

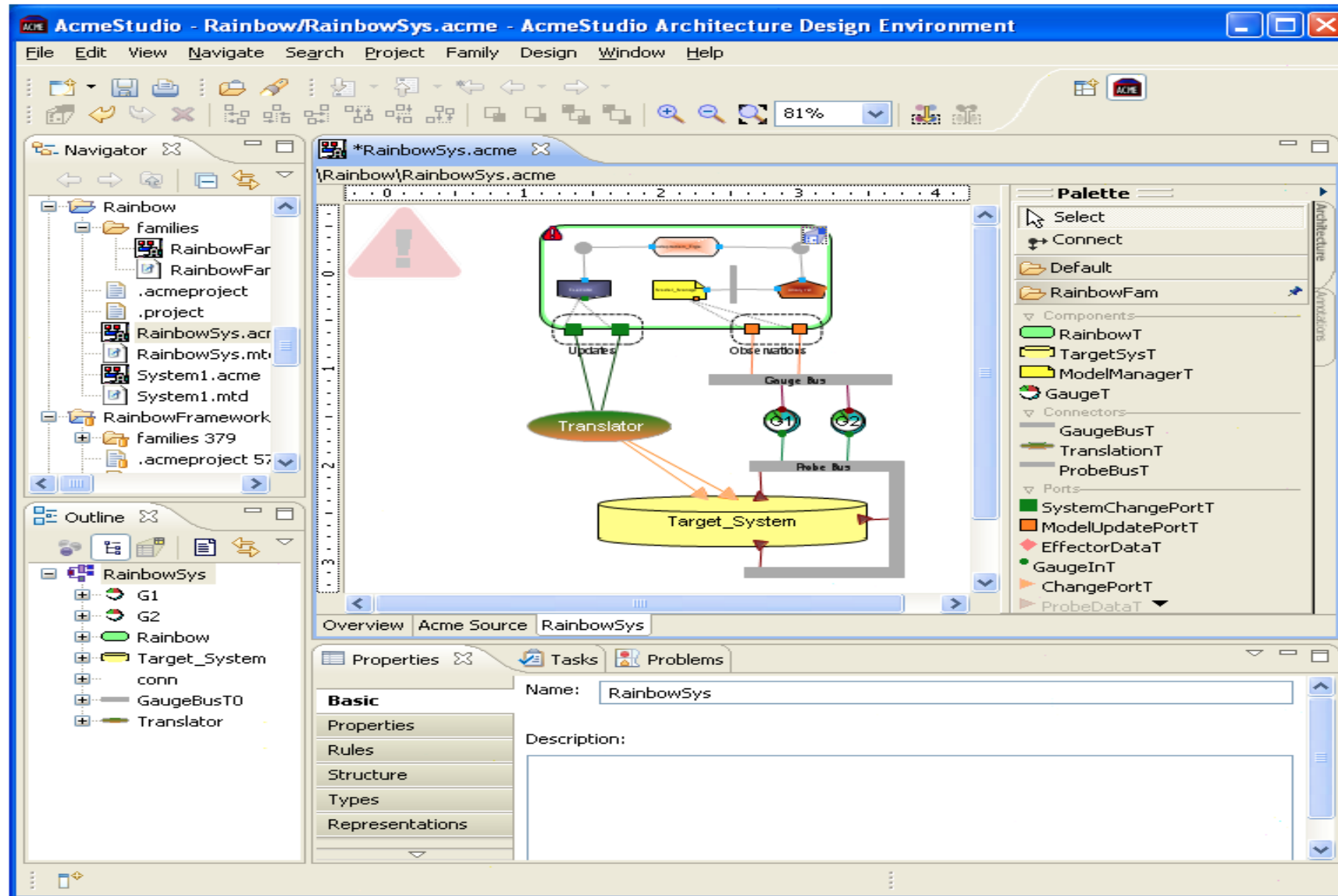
- ACME стартира с цел да може да се използва за обмяна на архитектурни описания между различни инструменти за архитектурно проектиране.
- Сравнително малък и доста лесен за използване език. Основните му елементи са: *система*, *компонент*, *порт*, *конектор* и *роля*.
- Предоставя механизмите за адекватно описание на структурата на дадена система, но когато става въпрос за други нейни аспекти като поведение на системата или динамична реконфигурация, ACME разчита на разширения (напр. Plastik) или други езици.

ADLs - ACME



```
1. System simple_cs = {
2.   Component client = {
3.     Port send-request;
4.     Property Aesop-style : style-id = client-server;
5.     Property UniCon-style : style-id = client-server;
6.     Property source-code : external = "CODE-LIB/client.c";
7.   }
8.   Component server = {
9.     Port receive-request;
10.    Property idempotence : boolean = true;
11.    Property max-concurrent-clients : integer = 1;
12.    source-code : external = "CODE-LIB/server.c";
13.  }
14.  Connector rpc = {
15.    Role caller;
16.    Role callee;
17.    Property asynchronous : boolean = true;
18.    max-roles : integer = 2;
19.    protocol : Wright = " ... ";
20.  }
21.  Attachment client.send-request to rpc.caller;
22.  Attachment server.receive-request to rpc.callee;
23. }
```

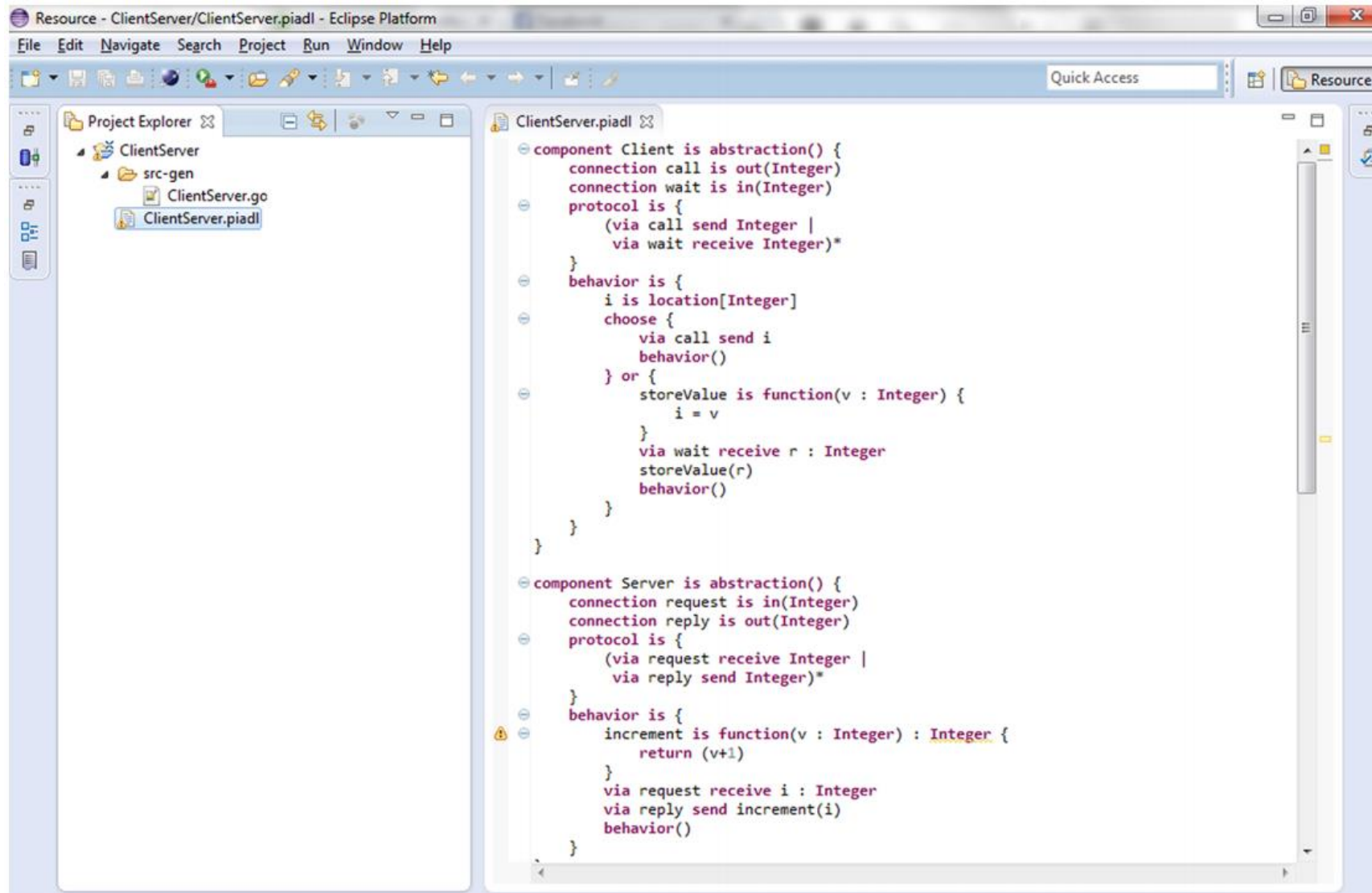
ADLs - ACME



ADLS - π -ADL

- π -ADL е формален ADL базиран на π -calculus, фокусиран върху динамичната перспектива и адресирането на динамични реконфигурации.
- Той поддържа (в зависимост от използването на инструменти или други езици) както предвидени, така и непредвидени динамични реконфигурации на инстанции.
- Има разработени различни инструменти, пр. редактор. Също така има разработен транслатор, с който може да се стигне от π -ADL спецификация до програмен код на GO.

ADLS - π -ADL

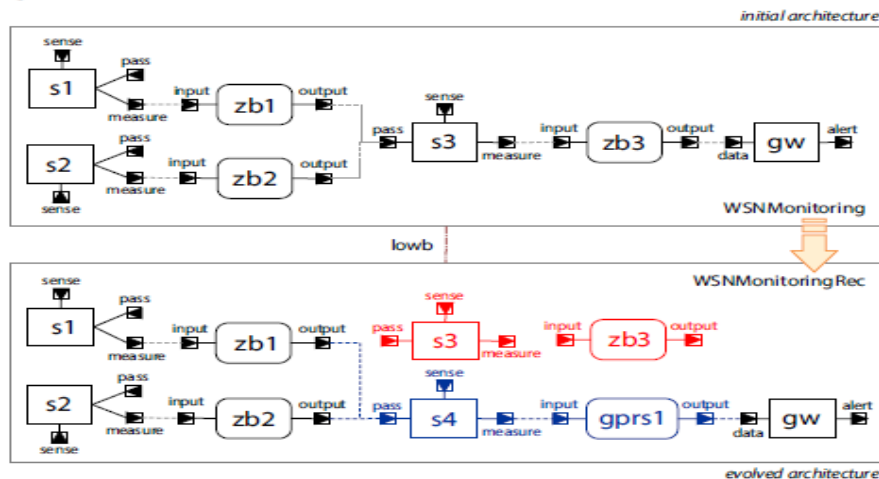


ADLS - π -ADL

```

architecture WSNMonitoringRec is
  abstraction(lowb : connection[Boolean], iarch : Any) {
    behavior is {
      abs is sequence[Behavior]
      abs = decompose iarch      // decomposing WSNMonitoring
      compose {
        s1 is abs[0]              // previous Sensor instance
        and s2 is abs[1]          // previous Sensor instance
        and s4 is Sensor()        // new Sensor instance
        and zb1 is abs[3]         // previous ZigBee instance
        and zb2 is abs[4]         // previous ZigBee instance
        and gprs1 is GPRS()       // new GPRS instance
        and gw is abs[6]          // previous Gateway instance
      } where {
        s1::measure unifies zb1::input
        s2::measure unifies zb2::input
        zb1::output unifies s4::pass
        zb2::output unifies s4::pass
        s4::measure unifies gprs1::input
        gprs1::output unifies gw::data
      }
    }
  }
  behavior is {                  // controlling behavior
    connection lowb is in(Boolean)
    iarch = WSNMonitoring(lowb) // initial architecture
    via lowb receive v : Boolean
    if (v == true) then {        // low battery notification
      WSNMonitoringReconf(iarch)
    }
  }
}

```



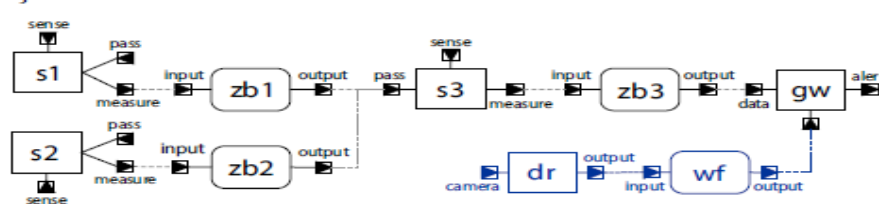
```

component UAV is abstraction() {
  unobservable
}

connector WiFi is abstraction() {
  unobservable
}

component Gateway is abstraction() {
  connection data is in(Integer)
  connection image is in(Any)
  behavior is {
    triggerAlert is function(measure : Integer) : String {
      unobservable
    }
    processImage is function(i : Any) : Boolean {
      unobservable
    }
  }
  via data receive d : Integer
  risk is location[String]
  risk = triggerAlert(d)
  if (risk == "High" || risk == "Very high") then {
    compose {
      dr is UAV()           // UAV (drone) component instance
      and wf is WiFi()      // WiFi connector instance
    } where {
      dr::output unifies wf::input
      wf::output unifies self::image
    }
    via image receive i : Any
    if (processImage(i) == true) then {
      via alert send "Flood risk confirmed"
    }
  }
}
  via alert send risk
}

```



Supporting Dynamic Software Architectures: From Architectural Description to Implementation, E. Cavalcante et al.

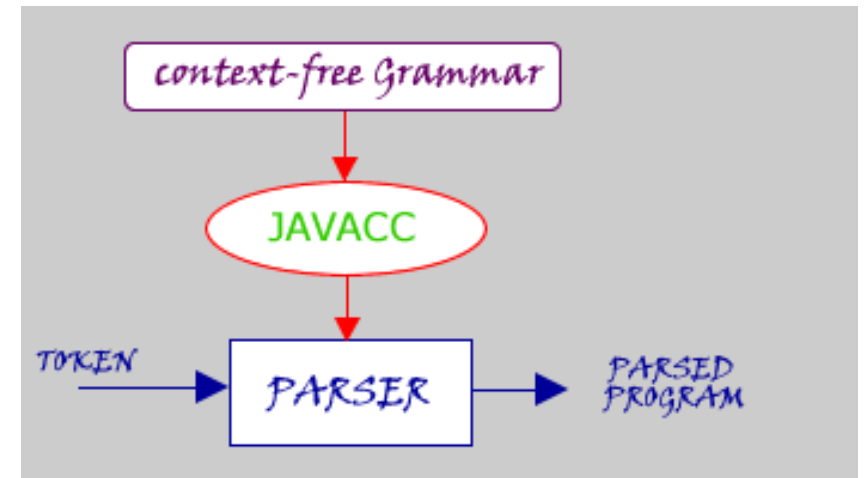
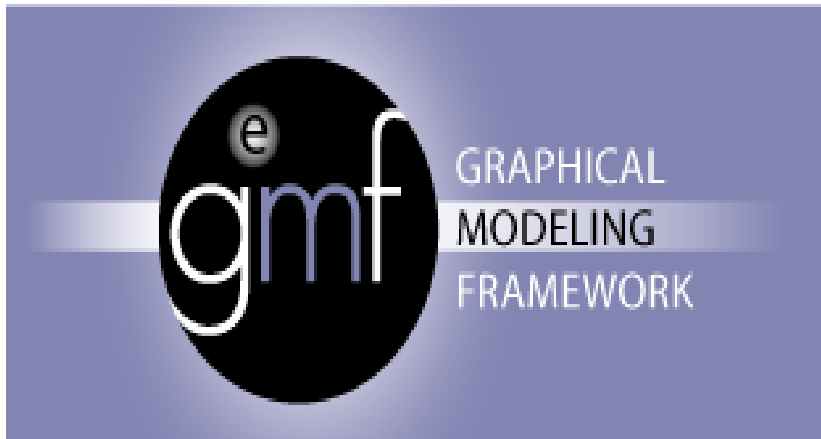
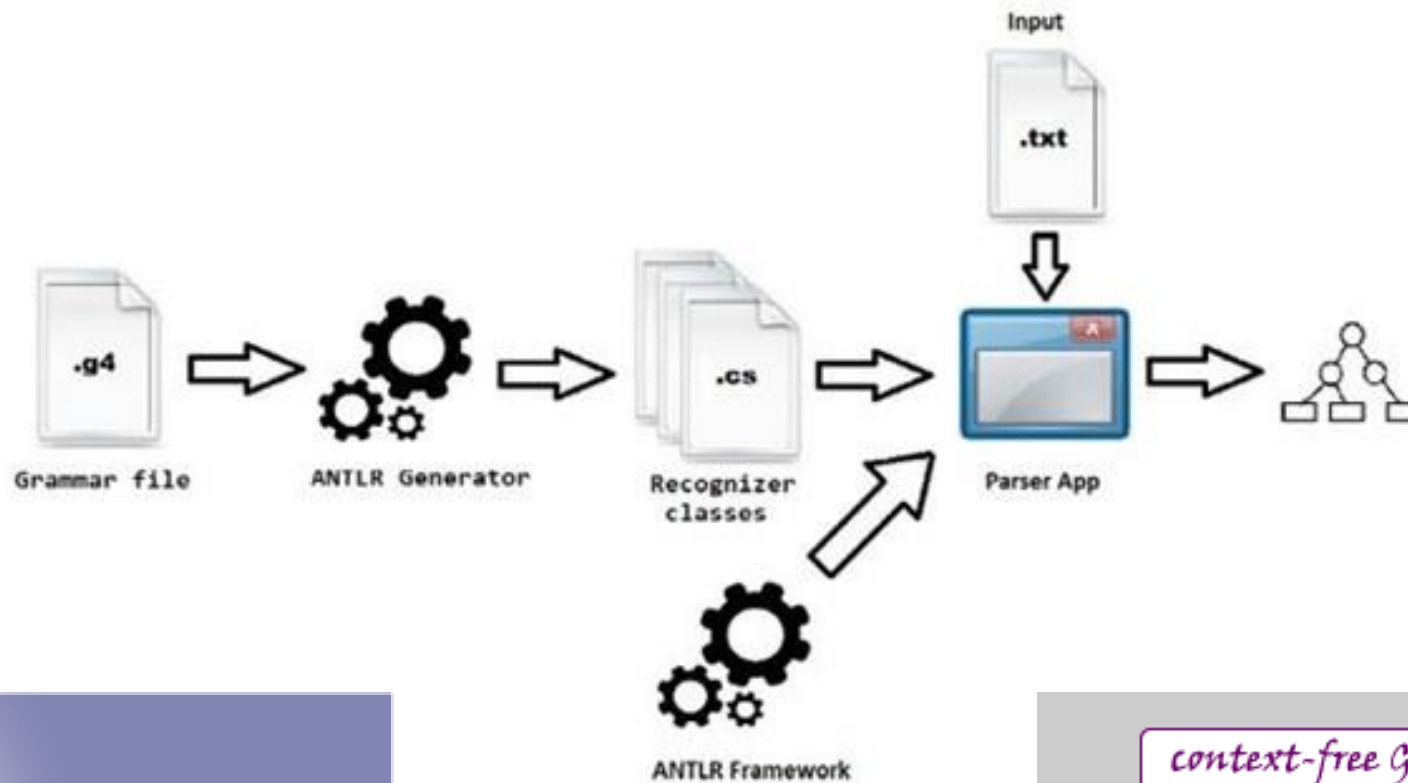
Architecture Description Languages

- ***Darwin*** – един от първите ADLs. Той следва компонентно-базиран подход и се фокусира върху разпределени (distributed) приложения.
- ***Wright*** – фокусиран предимно върху спецификацията и анализа на взаимодействията между компонентите. Създадено е и разширение (Dynamic Wright) за по-добра поддръжка на динамични реконфигурации.
- ***Koala*** – фокусира се върху описанието на архитектури на софтуера в продуктите свързани с електроника. Ограничени средства за уточняване на поведението.
- ***Rapide*** – компонентно-базиран ADL, който позволява да се симулират архитектурни описания и предоставя инструментите за анализ на резултатите от тези симулации.

Architecture Description Languages

- **AADL** – ADL фокусиран върху спецификацията и анализа на вградени (embedded) системи с критична производителност, който съдържа конструкции за моделиране както на софтуерни, така и на хардуерни компоненти. Също така поддържа значителен брой от инструменти.
- **xADL** – един xml-базиран ADL с големи възможности за разширяване и гъвкавост. Предоставя както текстови, така и графични изображения на архитектурата и е важно че могат да се използват много съществуващи xml инструменти.
- **C2** – поддържа описанието на системи използвайки базиран на събития стил.
- ...

Developing an ADL



Developing an ADL - *Xtext*

- *Xtext* is a framework for defining both *GPLs* and *DSLs* with a “*full infrastructure*” – parser, editor, compiler etc.
- Building DSLs with the Xtext Eclipse plugin offers a number of features that can be used after the definition of the grammar.
- In combination with *Xtend* writing a compiler becomes easier.



Xtext - Grammar definition

keywords

single assignment

multivalued assignment

cardinality

references

```
grammar org.xtext.example.mydsl :components //////////////////////////////////////
generate jadl "http://www.example.org/jadl"

import "http://www.example.org/xtext-examples.xtext"
import "http://www.example.org/xtext-examples.xtext"
import "http://www.example.org/xtext-examples.xtext"

Model :
    types+=typeDeclaration
    ;

typeDeclaration :
    interfaces+=interfaceDeclaration
    components+=componentDeclaration
    connectors+=connectorDeclaration
    archs+=architecture
    stmts+=XJSStatement
    ;

componentDeclaration :
    'component' name=ValidID
    componentBodyDecl=componentBody
    ;

componentBody :
    {componentBody} '{' componentBodyDeclarationDecl+=componentBodyDeclaration+
    '}'
    ;

componentBodyDeclaration :
    portDeclaration
    | attributeDeclaration
    | configPortDeclaration
    | compStatement=XJSStatement
    ;

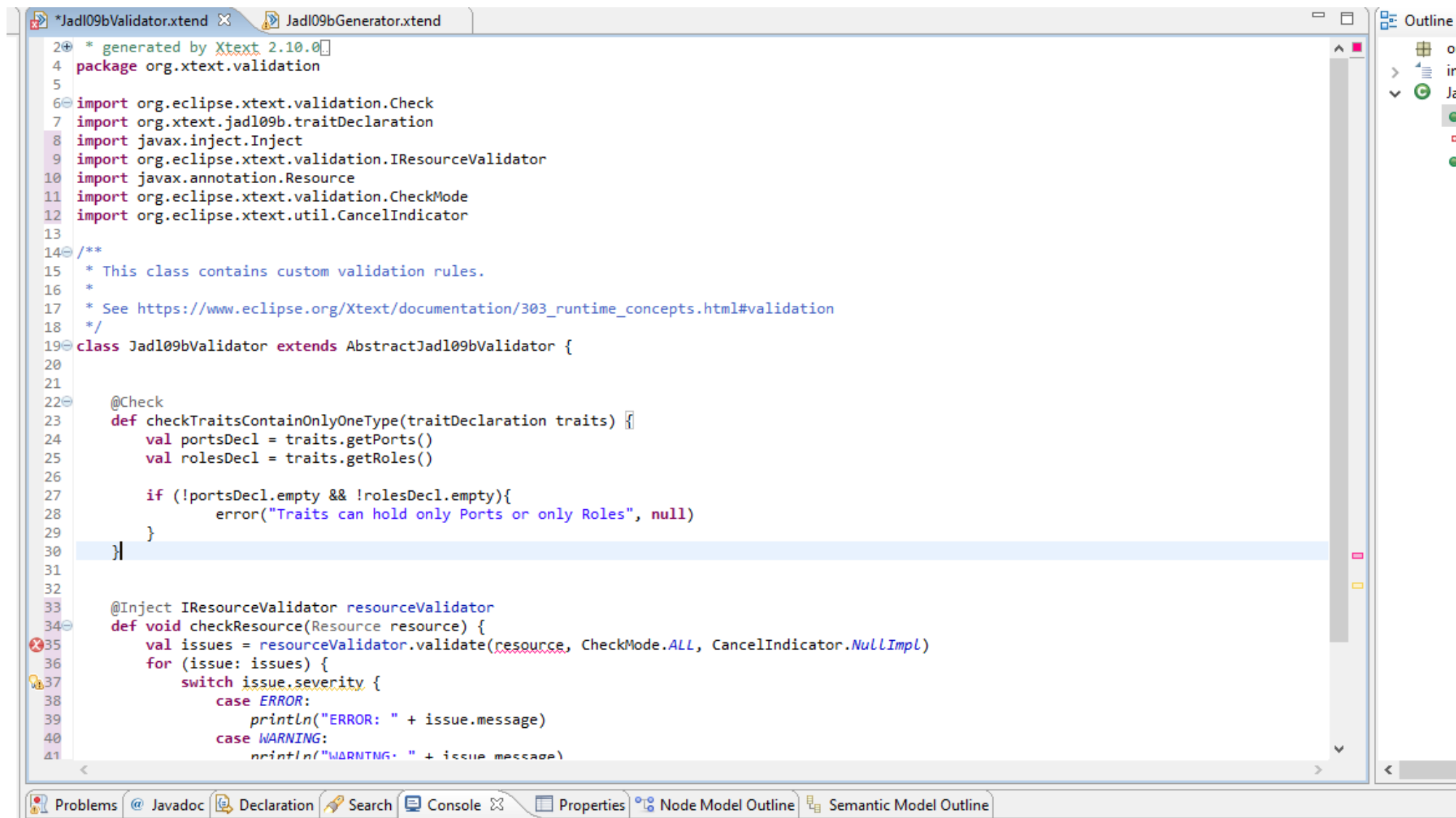
portDeclaration :
    {portDeclaration} type=portAndRoleType 'port' ('synchronized')?
    Interimpl=[interfaceDeclaration] name=ValidID ';'
    ;

portAndRoleType :
    "provides"
    | "requires"
    ;

attributeDeclaration :
    'attribute' attrType=types name=ValidID '=' attrInit=(ValidID
    | STRING
    | ("true" | "false"))
    ;

configPortDeclaration :
    'config' portToConnect=ValidID
    configmemberDeclDecl+=configMemberDeclaration
    ;
```

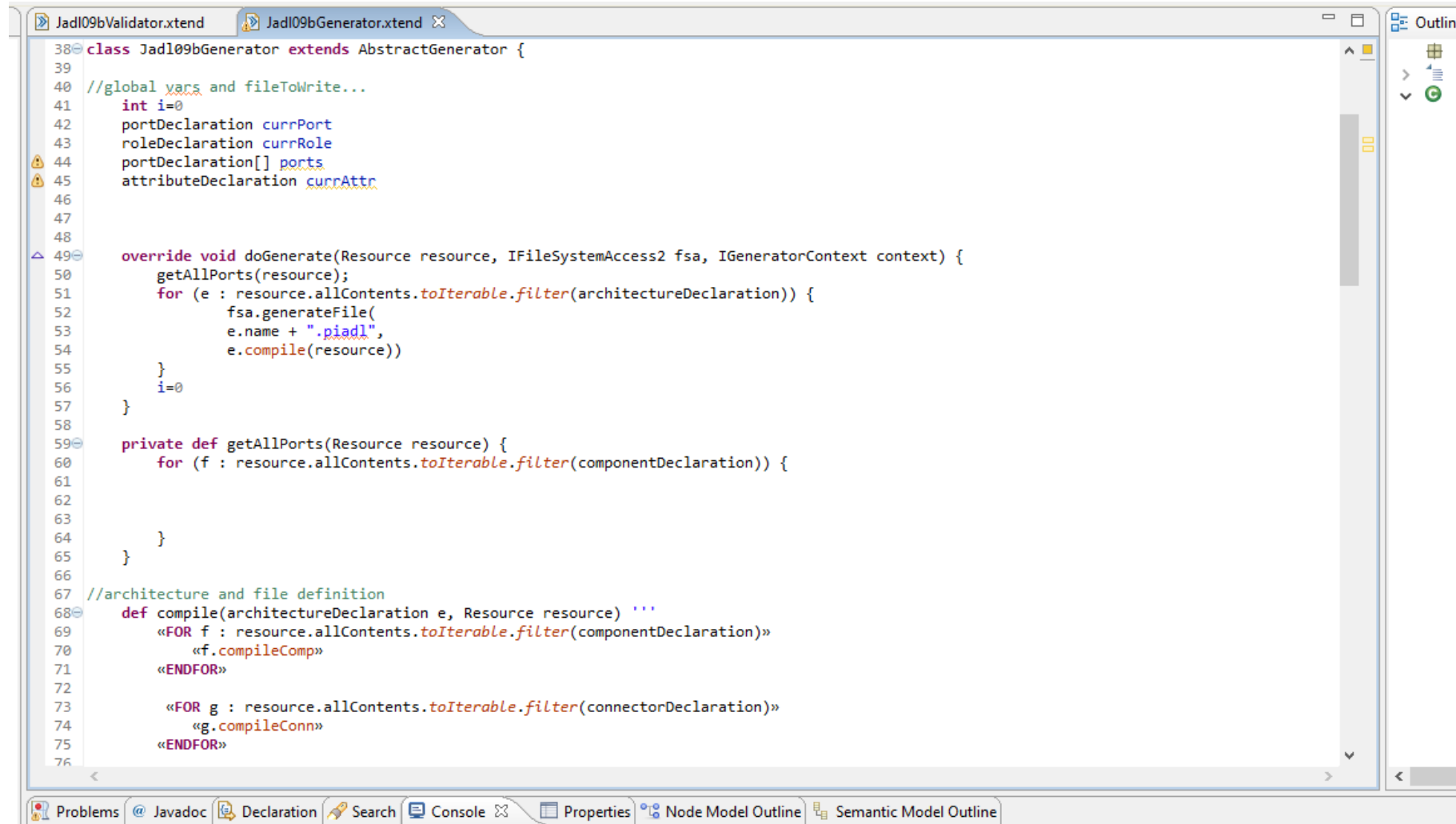

Xtext - Validator definition



```
*Jadl09bValidator.xtend
Jadl09bGenerator.xtend

2+ * generated by Xtext 2.10.0
4 package org.xtext.validation
5
6 import org.eclipse.xtext.validation.Check
7 import org.xtext.jadl09b.traitDeclaration
8 import javax.inject.Inject
9 import org.eclipse.xtext.validation.IResourceValidator
10 import javax.annotation.Resource
11 import org.eclipse.xtext.validation.CheckMode
12 import org.eclipse.xtext.util.CancelIndicator
13
14 /**
15  * This class contains custom validation rules.
16  *
17  * See https://www.eclipse.org/Xtext/documentation/303\_runtime\_concepts.html#validation
18  */
19 class Jadl09bValidator extends AbstractJadl09bValidator {
20
21
22 @Check
23 def checkTraitsContainOnlyOneType(traitDeclaration traits) {
24     val portsDecl = traits.getPorts()
25     val rolesDecl = traits.getRoles()
26
27     if (!portsDecl.empty && !rolesDecl.empty){
28         error("Traits can hold only Ports or only Roles", null)
29     }
30 }
31
32
33 @Inject IResourceValidator resourceValidator
34 def void checkResource(Resource resource) {
35     val issues = resourceValidator.validate(resource, CheckMode.ALL, CancelIndicator.NullImpl)
36     for (issue: issues) {
37         switch issue.severity {
38             case ERROR:
39                 println("ERROR: " + issue.message)
40             case WARNING:
41                 println("WARNING: " + issue.message)
```

Xtext - Generator definition



The screenshot shows an IDE window with two tabs: 'Jadl09bValidator.xtend' and 'Jadl09bGenerator.xtend'. The 'Jadl09bGenerator.xtend' tab is active, displaying the following code:

```
38 class Jadl09bGenerator extends AbstractGenerator {
39
40 //global vars and fileToWrite...
41 int i=0
42 portDeclaration currPort
43 roleDeclaration currRole
44 portDeclaration[] ports
45 attributeDeclaration currAttr
46
47
48
49 override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
50     getAllPorts(resource);
51     for (e : resource.allContents.toIterable.filter(architectureDeclaration)) {
52         fsa.generateFile(
53             e.name + ".piadl",
54             e.compile(resource))
55     }
56     i=0
57 }
58
59 private def getAllPorts(Resource resource) {
60     for (f : resource.allContents.toIterable.filter(componentDeclaration)) {
61
62
63     }
64 }
65 }
66
67 //architecture and file definition
68 def compile(architectureDeclaration e, Resource resource) '''
69     «FOR f : resource.allContents.toIterable.filter(componentDeclaration)»
70     «f.compileComp»
71     «ENDFOR»
72
73     «FOR g : resource.allContents.toIterable.filter(connectorDeclaration)»
74     «g.compileConn»
75     «ENDFOR»
76
```

The IDE interface includes a right-hand 'Outline' pane showing a tree view of the project structure with nodes like 'or', 'im', and 'Jadl'. The bottom status bar contains tabs for 'Problems', 'Javadoc', 'Declaration', 'Search', 'Console', 'Properties', 'Node Model Outline', and 'Semantic Model Outline'.

Xtext - Grammar compilation

```
Model :
    types+=typeDeclaration*
;

typeDeclaration :
    interfaces+=interfaceDeclaration
    | components+=componentDeclaration
    | connectors+=connectorDeclaration
    | archs+=architectureDeclaration
    | stmnts+=XJSStatementOrBlock
;

componentDeclaration :
    'component' name=ValidID
    componentBodyDecl=componentBody
;

componentBody :
    {componentBody} '{' componentBodyDeclarationDecl+=componentBodyDeclaration+ '}'
;

componentBodyDeclaration :
    portDeclaration
    | attributeDeclaration
    | configPortDeclaration
    | compStatement=XJSSingleStatement
;

portDeclaration :
    {portDeclaration} type=portAndRoleType 'port' ('synchronized')? Interimpl=[interfaceDeclaration] name=ValidID ';'
;

portAndRoleType :
    "provides"
    | "requires"
;

attributeDeclaration :
    'attribute' attrType=types name=ValidID '=' attrInit=(ValidID
    | STRING
    | ("true" | "false")
    | INTEGER
    | FLOAT) ';'
;

configPortDeclaration :
    'config' portToConfig=[portDeclaration] 'as' '{'
    configmemberDeclDecl+=configmemberDecl+
    '}'
;
```

```
component Server {

    provides port IProcess req;
    requires port IResponse reply;

    attribute int curLoad = 0;

    attribute string threadExec = "single";

    config req as {
        service void procRequest(type data){
            //create response and reply
            type resp = processReq(data);
            reply.aResponse(resp);
        }
    }

    while (true) {
        select {
            when (curLoad < maxNum) =>
                process;
        }
        or {
            when (curLoad == maxNum) =>
                delay_until curLoad < maxNum;
        }
    }
}
```

Xtext - Grammar compilation

```
Model :  
    types+=typeDeclaration*  
    ;
```

```
typeDeclaration :  
    interfaces+=interfaceDeclaration  
    | components+=componentDeclaration  
    | connectors+=connectorDeclaration  
    | archs+=architectureDeclaration  
    | stmnts+=XJSStatementOrBlock  
    ;
```

```
componentDeclaration :  
    'component' name=ValidID  
    componentBodyDecl=componentBody  
    ;
```

```
componentBody :  
    {componentBody} '{' componentBodyDeclarationDecl+=componentBodyDeclaration+  
    ;
```

```
componentBodyDeclaration :  
    portDeclaration  
    | attributeDeclaration  
    | configPortDeclaration  
    | compStatement=XJSSingleStatement  
    ;
```

```
portDeclaration :  
    {portDeclaration} type=portAndRoleType 'port' ('synchronized')? Interimpl=[interfaceDeclaration] name=ValidID ';' ;
```

```
portAndRoleType :  
    "provides"  
    | "requires"  
    ;
```

```
attributeDeclaration :  
    'attribute' attrType=types name=ValidID '=' attrInit=(ValidID  
    | STRING  
    | ("true" | "false")  
    | INTEGER  
    | FLOAT) ';' ;
```

```
configPortDeclaration :  
    'config' portToConfig=[portDeclaration] 'as' '{'  
    configmemberDeclDecl+=configmemberDecl+  
    '}'  
    ;
```

```
component Server {
```

```
    provides port IProcess req;  
    requires port IResponse reply;
```

```
    attribute int curLoad = 0;
```

```
    attribute string threadExec = "single";
```

```
    config req as {  
        service void procRequest(type data){  
            //create response and reply  
            type resp = processReq(data);  
            reply.aResponse(resp);  
        }  
    }
```

```
    while (true) {  
        select {  
            when (curLoad < maxNum) =>  
                process;;  
            or {  
                when (curLoad == maxNum) =>  
                    delay_until curLoad < maxNum; }  
        }  
        end;  
    }
```

```
}
```

Generated text editor

```
component Client {
```

```
  requires port IRequest send;  
  provides port IReceive wait;
```

```
  config w
```

wait - Client.wait

Ctrl+Space to show shortest proposals

```
interface IReceive {  
  service void Received(type data);  
}
```

```
component Client {
```

```
  provides port IReceive2 wait;
```

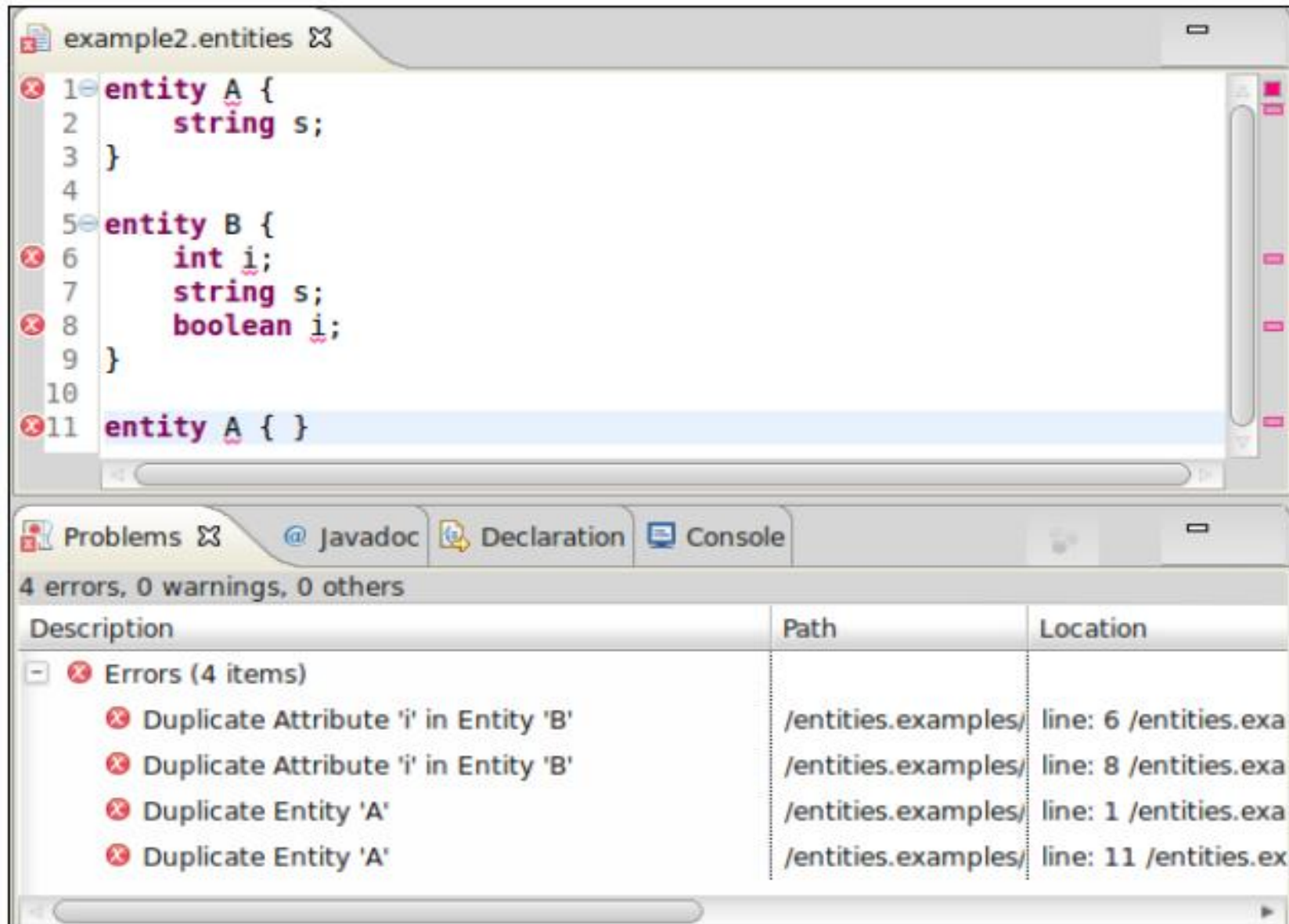
 IReceive2 cannot be resolved.

1 quick fix available:

 [Change to 'IReceive'](#)

Press 'F2' for focus

Xtext - Validation



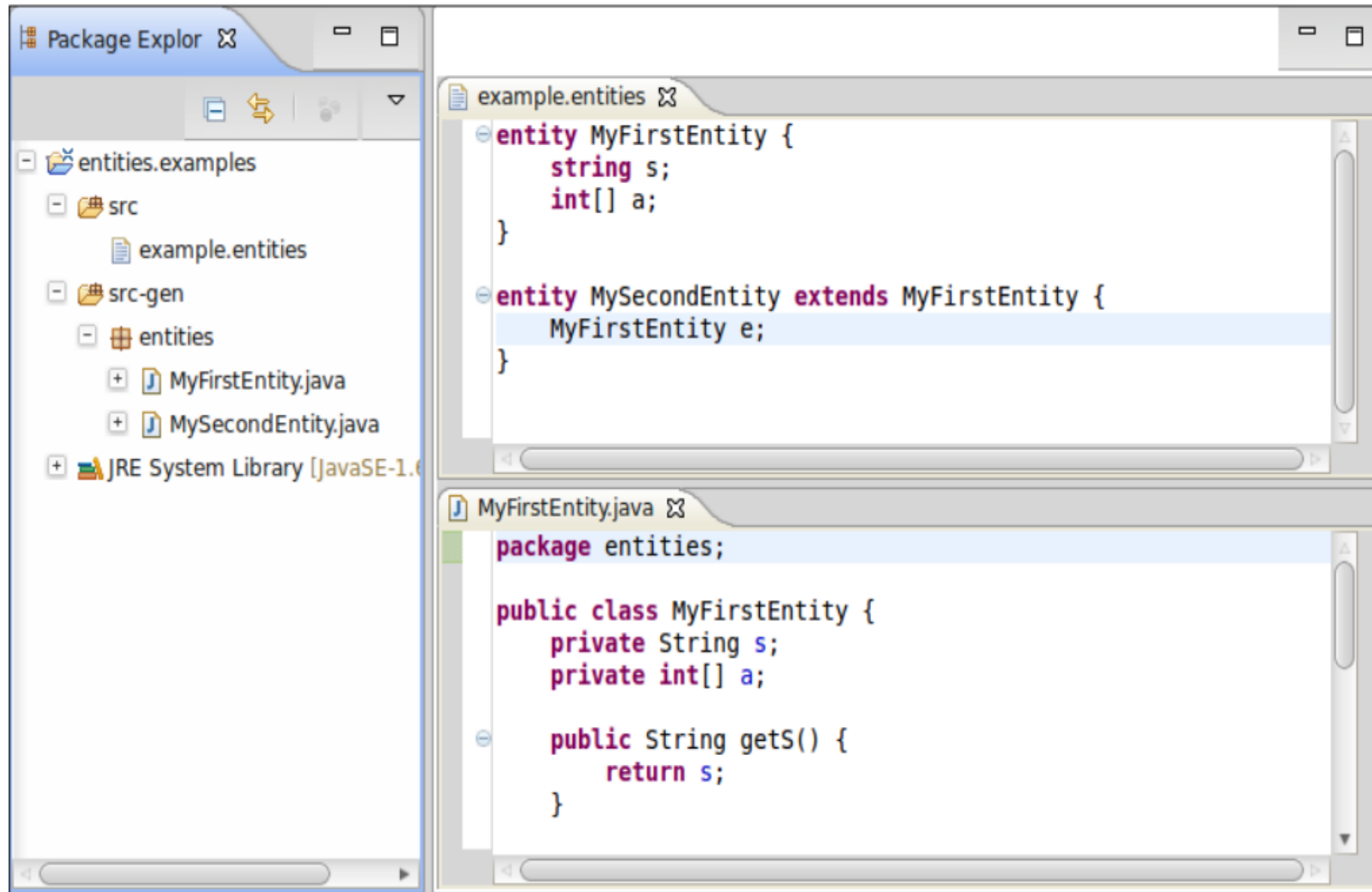
The screenshot shows an Xtext editor window titled "example2.entities". The editor contains the following code:

```
1 entity A {  
2     string s;  
3 }  
4  
5 entity B {  
6     int i;  
7     string s;  
8     boolean i;  
9 }  
10  
11 entity A { }
```

Four red 'X' error markers are visible on the left margin, corresponding to lines 1, 5, 6, and 8. Below the editor is a "Problems" panel with tabs for "Problems", "Javadoc", "Declaration", and "Console". The "Problems" tab is active, showing "4 errors, 0 warnings, 0 others". The error list is as follows:

Description	Path	Location
Errors (4 items)		
✗ Duplicate Attribute 'i' in Entity 'B'	/entities.examples/	line: 6 /entities.exa
✗ Duplicate Attribute 'i' in Entity 'B'	/entities.examples/	line: 8 /entities.exa
✗ Duplicate Entity 'A'	/entities.examples/	line: 1 /entities.exa
✗ Duplicate Entity 'A'	/entities.examples/	line: 11 /entities.ex

Xtext - Code Generation



Xtext - Adding tools

Node Model Outline

- > [RuleCall] interfaceDeclaration
- ▼ [RuleCall] typeDeclaration
 - ▼ [RuleCall] interfaceDeclaration
 - 👁 [TerminalRule] WS
 - 🌿 [Keyword] interface
 - ▼ [RuleCall] ValidID
 - 👁 [TerminalRule] WS
 - 🌿 [RuleCall] ID
 - ▼ [RuleCall] interfaceBody
 - 👁 [TerminalRule] WS
 - 🌿 [Keyword] {
 - > [RuleCall] interfaceBodyDeclaration
 - 👁 [TerminalRule] WS
 - 🌿 [Keyword] }
- ▼ [RuleCall] typeDeclaration
 - ▼ [RuleCall] componentDeclaration
 - 👁 [TerminalRule] WS
 - 🌿 [Keyword] component
 - ▼ [RuleCall] ValidID
 - 👁 [TerminalRule] WS
 - 🌿 [RuleCall] ID
 - ▼ [Action] componentBody
 - > [RuleCall] componentBody
 - 👁 [TerminalRule] WS
 - 🌿 [Keyword] {
 - > [RuleCall] componentBodyDeclaration
 - > [RuleCall] componentBodyDeclaration
 - > [RuleCall] componentBodyDeclaration
 - 👁 [TerminalRule] WS
 - 🌿 [Keyword] }
 - ▼ [RuleCall] typeDeclaration
 - > [RuleCall] interfaceDeclaration

Semantic Model Outline

- ▼ [Model]
 - URI platform:/resource/LBserver/src/archtest.jadl#/0
 - ▼ types (6)
 - > [typeDeclaration]
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.1
 - ▼ interfaces (1)
 - > [interfaceDeclaration] IReceive
 - ▼ [typeDeclaration]
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.2
 - ▼ components (1)
 - ▼ [componentDeclaration] Client
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.2/@components.0
 - name = Client
 - ▼ componentBodyDecl = [componentBody]
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.2/@components.0/@componentBodyDecl
 - ▼ componentBodyDeclarationDecl (3)
 - ▼ [portDeclaration] send
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.2/@components.0/@componentBodyDecl/@componentBodyDeclarationDecl.0
 - type = requires
 - ↳ Interimpl -> [interfaceDeclaration] IRequest
 - name = send
 - > [portDeclaration] wait
 - ▼ [configPortDeclaration]
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.2/@components.0/@componentBodyDecl/@componentBodyDeclarationDecl.2
 - ↳ portToConfig -> [portDeclaration] wait
 - > configmemberDeclDecl (1)
 - ▼ [typeDeclaration]
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.3
 - ▼ interfaces (1)
 - ▼ [interfaceDeclaration] IResponse
 - URI platform:/resource/LBserver/src/archtest.jadl#/0/@types.3/@interfaces.0